**数据结构课程设计报告**

**班级：2014211302 学号：2014211168 周尧棋**

**班级：2014211302 学号：2014211175 加苏尔·吐尔地**

**班级：2014211303 学号：2014211202 廖竞鑫**

**一、设计任务的描述**

城市之间有三种交通工具（汽车、火车和飞机）相连，某旅客于某一时刻向系统提出旅行要求，系统根据该旅客的要求为其设计一条旅行线路并输出；系统能查询当前时刻旅客所处的地点和状态（停留城市/所在交通工具）。并且支持多用户订票，多用户同时查询。

**二、功能需求说明及分析**

我们所选择建立交通连接的城市是北京，上海，广州，郑州，长沙，成都，重庆，乌鲁木齐，昆明，沈阳，西安，武汉，这12个城市。

我们建立了每两个城市之间各种交通工具的时刻表，其中出发、到达时间和费用等信息都来自真实的出行时刻表，一条一条摘自于网上。时刻表中的路线分别有途经城市和直达两种出行路线，总计有几百条，基本能将这12个城市之间的出行路线覆盖。

用户进入系统后首先可以选择键盘输入和图形化界面两种模式，进入选择界面后选择“添加旅客”功能后可进行购票操作。输入用户个人姓名和ID号，然后输入用户的起点、终点、途径城市、期待出发的时间和选择的旅行策略，旅行策略有：

最少费用策略：无时间限制，费用最少即可

最少时间策略：无费用限制，时间最少即可

限时最少费用策略：在规定的时间内所需费用最省

旅行模拟查询系统以时间为轴向前推移，每10秒左右向前推进1个小时(非查询状态的请求不计时)，制作的系统时间精确到小时。

在选择界面选择“查询状态”功能后首先输入查询的旅客的用户信息，然后可以清楚的显示被查询者当前的状态，系统能够实现多用户状态同时查询。

在出行过程中不考虑城市内换乘交通工具所需时间，但考虑了到达某一城市后等待下一趟班次的停留时间。

建立了日志文件，对旅客状态变化和键入等信息进行了记录。

**三、总体方案设计说明**

**1.软件开发环境**：Dev-C++

**2.模块划分：**

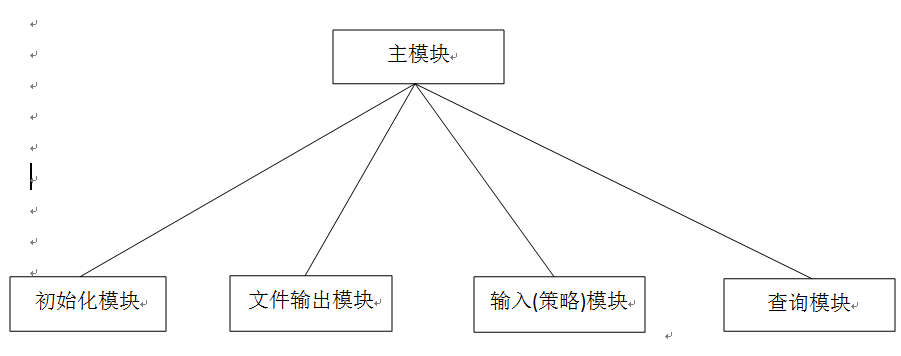
键盘输入版本与图形化版本的基本模块划分如下图，主模块下分为初始化模块、文件输出、输入（策略）模块和查询模块四个模块。

初始化模块进行各种初始化操作，主要是从文件中读取线路存入路线图的操作，以及建立用户链表头、打印链表头等。

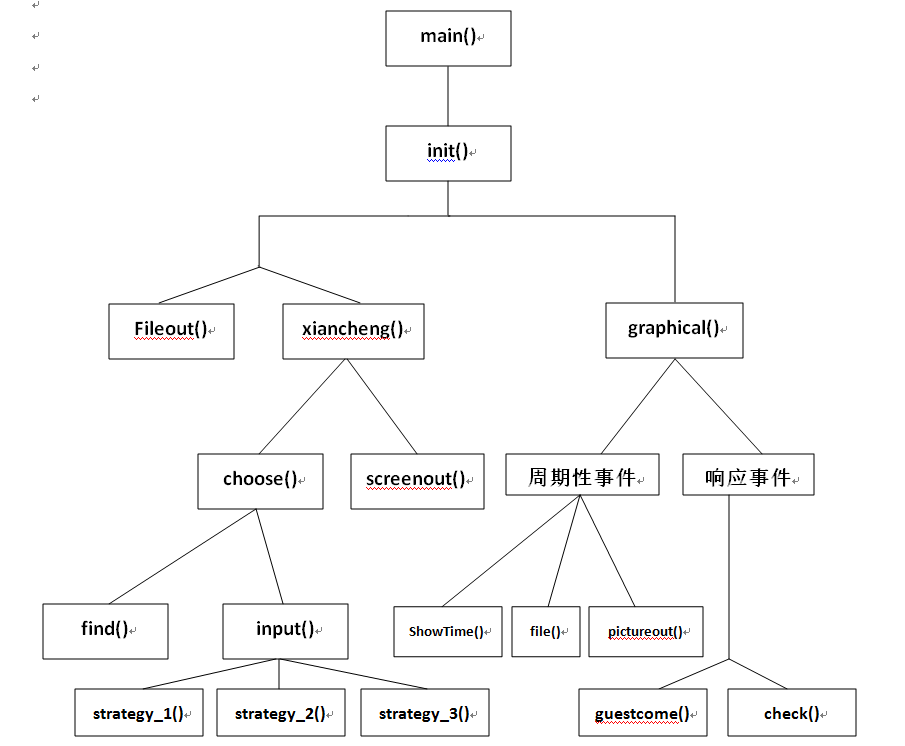
输入（策略）模块接收键盘输入命令，分析命令并调用相应的策略模块，并将给用户设计的策略输出。

查询模块接收键盘输入命令后可显示旅客的当前状态。

文件输出模块输出每个时刻的事件至日志文件"output.txt"中。



**3.总体结构：**



main()函数中有初始化函数，键盘版本函数，图形化函数。

进入main()函数后先运行初始化函数init()，然后根据输入选择程序运行方式。输入1进入键盘输入模式，输入2进入图形化界面。

**初始化函数：**

1. 首先运行 Creategraph()函数创建城市路线图graph G，从文件中读取数据，将每个城市之间的不同交通工具的线路存入城市路线图G中。

(2)再运行cinit()函数将用户链表和打印链表初始化。

**键盘模块：**

键盘模块下有Fileout()文件输出模块和xiancheng()线程模块

Fileout()文件输出模块每小时将发生的事件输入到日志文件"output.txt"中。

xiancheng()线程模块分为screenout()和choose()两大模块函数，choose()线程函数根据输入调用input()函数（添加客户）和find()函数（查询状态），input()函数中根据用户的输入调用三个策略函数strategy\_1() 、strategy\_2() 、strategy\_3()。若find()函数查询成功，则screenout()函数将每小时客户的状态输出到屏幕上。

**图形化模块：**

根据定时器响应周期性事件或者按键按下事件。

在周期性事件中，ShowTime()函数每过一小时（系统时间10s为一小时）在界面上显示当前时间，file()函数则是每小时向文件输出所有旅客的当前状态。pictureout()函数每小时在图形化界面上输出旅客当前状态。

在按键按下事件中，则对鼠标点击或者键盘输入进行响应，点击“添加旅客”图标后，guestcome()函数进行添加旅客操作，点击“查询状态”图标后，check()函数对旅客进行查询判断，然后在周期性事件中输出旅客当前状态。点击“关闭”图标后关闭图形界面。

**四、数据结构说明和数据字典**

主要的数据结构说明：

typedef struct transportation

{

int start\_hour ; //每条路线的出发时间

int arrive\_hour ; //每条路线的到达时间

char Number[20] ; //交通工具的编号

int price ; //价格

int time ; //历经时间

struct transportation \*nextptr ;

} transportation,\*translist; //每种交通工具的路线链表

struct chengshi

{

translist trans[6]; //三种交通工具的链表

};

typedef struct

{

int city\_num; //城市数量

struct chengshi cs[12][12] ; //城市邻接矩阵

string city\_name[12] ; //城市名数组

string transportation\_name[3]; //交通工具名数组

} graph; //城市路线信息图

typedef struct route

{

int from ; //出发城市

int to ; //到达城市

int transportation ; //使用交通工具

int start\_day ; //出发天数

int start\_hour ; //出发小时数

int arrive\_hour ; //到达小时数

char Number[20] ; //交通工具的编号

int time ; //经历时间

int price ; //票价

struct route \*nextptr ;

} route, \*routelist; //策略路线存储结构

typedef struct Client

{

string Name ; //用户的姓名

string ID ; //用户的ID号

int city\_start ; //输入出发城市代号0-11

int city\_end ; //输入目的城市代号0-11

int pass\_city[12] ; //输入希望途径的城市

int city\_num ; //经过城市总数

int expected\_day ; //输入希望第几天出发

int expected\_hour ; //输入希望第几小时出发，

int strategy ; //用户希望的策略

int buy\_day ; //订票时第几天

int buy\_hour ; //订票时的小时数

int limit\_time ; //策略3限定的时间

int sco; //打印的flag

int scp[12];

routelist R ; //用户路线链表头结点

struct Client \*next ;

} Client, \*clientlist; //用户链表存储结构

struct temp

{

int tran ; //交通工具

int start\_hour ; //每种路线的出发时间

int arrive\_hour ; //每种路线的到达时间

char Number[20] ; //交通工具的编号

int time ; //总时间

int price ; //价格

} ; //暂时存储路线的结构体

**五、各模块设计说明**

**1.初始化模块**

(1)首先运行 Creategraph()函数创建城市路线图graph G，从文件一行一行地读取数据，将每个城市之间的不同交通工具的线路存入城市路线图G中。

(2)再运行cinit()函数将用户链表和打印链表初始化。

(3)初始化模块将所有线路信息存入图结构中，为其他模块奠定了数据基础。

**2.键盘模式操作模块**

**Fileout()文件输出模块：**

(1)将fileout()文件输出函数包装在Fileout()的一个线程里面，然后在启动键盘模块时不断运行Fileout线程。

1. 每隔一小时(系统时间10s为一小时)向文件进行一次输出，将尚未到达目的地的旅客当前的状态输出到文件中。

**xiancheng()线程模块**：

分为screenout()和choose()两大模块函数

choose()线程函数是一个捕捉键盘输入的函数。输入G时运行input()函数可以进行购票，此时Fileout和screenout线程都会暂停，购票之后重新恢复运行。用户的ID和名字作为购票的判断信息，若旅客想重复购买必须输入正确ID和名字不然将无法重复购买。在用户输入完信息后，系统根据旅客要求，为其分配策略。购买成功后旅客放入旅客链表中。

三种策略函数分别存放在strategy\_1，strategy\_2，strategy\_3三个文件中。输入P时运find函数可以对旅客进行查找，若有该旅客，则将旅客信息放入打印链表中。输入Q关闭所有线程，并关闭该程序。

screenout()线程函数在第一个旅客购买之后，每个一小时对屏幕进行刷新并且输出打印链表中的旅客信息，若打印链表无旅客则不进行输出。打印时根据当前时间T分为5个状态，对旅客当前位置进行判断，若T大于该第一个城市出发时间，则显示出发。若大于最后一个城市到达时间，则已到达目的地。若T小于下个城市出发时间，则在这两个城市之间旅行。若T小于最后一个城市到后两个城市之间旅行。若T大于某个城市到达时间，且小于下个城市出发时间，则旅客正在等待下一班次。

**三个策略的设计：**

(1)strategy\_1()函数

strategy\_1为最少费用策略，使用弗洛伊德算法计算任意两个城市之间的最小费用，根据旅客的输入城市为旅客设计旅行路线。

a.算法思想

它从图的带权邻接矩阵cost出发，其基本思想是：如果vi到vj有弧，则从vi到vj存在一条长度为cost[i][j]的路径。该路径不一定是最短路径，尚需要进行n次试探。首先考虑路径（ vi,v1,vj）是否存在（即判断弧（vi,v1）和(v1,vj)是否存在）。如果存在，则比较其路径长度。取长度较短者为从vi到vj的中间顶点的序号不大于1的最短路径。假如在路径上在增加一个顶点v2,即如果（vi,…,v2）和（v2,…,vj）分别是当前找到的中间顶点的序号不大于1的最短路径，那么，vi,..,v2,…,vj就有可能是从vi到vj中间顶点的序号不大于2的最短路径。将它和已经得到的从vi到vj中间顶点的序号不大于1的最短路径相比较，从中选出中间序号不大于2的最短路径之后，再增加一个顶点v3,继续进行试探。依次类推，直到经过n次比较，最后求得的必是从vi到vj的最短路径。按此方法，可以同时求得各对顶点间的最短路径。

b.算法

从任意一条单边路径开始。所有两点之间的距离是边的权，如果两点之间没有边相连，则权为无穷大。

对于每一对顶点u和 v，看看是否存在一个顶点 w使得从u到w再到v比已知的路径更短。如果是更新它。

把图用邻接矩阵G表示出来，如果从Vi到Vj有路可达，则G[i,j]=d，d表示该路的长度；否则G[i,j]=无穷大。定义一个矩阵D用来记录所插入点的信息，D[i,j]表示从Vi到Vj需要经过的点，初始化D[i,j]=j。把各个顶点插入图中，比较插点后的距离与原来的距离，G[i,j]=min(G[i,j], G[i,k]+G[k,j])，如果G[i,j]的值变小，则D[i,j]=k。在G中包含有两点之间最短道路的信息，而在D中则包含了最短通路径的信息。

c.特点

Floyd算法适用于APSP(All Pairs Shortest Paths，多源最短路径)，是一种动态规划算法，稠密图效果最佳，边权可正可负。此算法简单有效，由于三重循环结构紧凑，对于稠密图，效率要高于执行|V|次[Dijkstra算法](http://baike.baidu.com/view/7839.htm)，也要高于执行V次[SPFA算法](http://baike.baidu.com/view/4700690.htm)。

优点：容易理解，可以算出任意两个节点之间的最短距离,代码编写简单。

缺点：时间复杂度比较高，为O，不适合计算大量数据。

**d.在strategy\_1()中首先初始化temp类型数组的D[12][12].price，为两城市间所有线路中的最少费用，并将其出发时间、到达时间、交通工具、总时间存入D[12][12]结构体数组中。再利用三层循环计算实际两城市间旅行的最短费用，根据旅客输入的途径城市给旅客分配路线，存入用户的routelist链表中。**

(2)strategy\_2()函数

strategy\_2为最少时间策略，使用分段的迪杰斯特拉算法根据用户输入的期待天数和期待出发时间为用户分配最短时间策略。

a.算法思想

设置并逐步扩充一个集合S，存放已求出其最短路径的顶点，则尚未确定最短路径的顶点集合是V-S，其中V为网中所有顶点集合。按最短路径长度递增的顺序逐个以V-S中的顶点加到S中．直到S中包含全部顶点，而V-S为空。

b.算法

设源点为vl，则S中只包含顶点vl，令W＝V-S，则W中包含除v1外图中所有顶点，vl对应的距离值为0，W中顶点对应的距离值是这样规定的：若图中有弧<vi,vj>，则vj顶点的距离为此弧权值，否则为∞ (一个很大的数)，然后每次从W中的顶点中选—个其距离值为最小的顶点vm加入到S中，每往S中加入一个顶点vm，就要对W中的各个顶点的距离值进行一次修改。若加进vm做中间顶点，使<vi,vm>+<vm,vj>的值小于<vi,vj>值，则用<vi,vm>+<vm,vj>代替原来vj的距离，修改后再在W中选距离值最小的顶点加入到S中，如此进行下去，直到S中包含了图中所有顶点为止。

下面以邻接矩阵存储来讨论迪杰斯特拉算法的具体实现。

为了找到从源点到其他顶点的最短路径，引入两个辅助数组dist[n],s[n],数组dist记录从源点到其他各顶点当前的最短距离，其初值为dist[i]=cost[v0][i], i=2,...,n.其中v0表示源点。从S之外的顶点集合V-S中选出一个顶点w,使dist[w]的值最小。于是从源点到达w只通过S中的顶点，把w加入集合S中调整dist中记录的从源点到V-S中每个顶点v的距离：从原来的dist[v]和dist[w]+cost[w][v]中选择较小的值作为新的dist[v],重复上述过程，直到S中包含V中其余各顶点的最短路径。

c.特点

迪杰斯特拉算法主要特点是以起始点为中心向外层层扩展，直到扩展到终点为止，为一种贪心算法，时间复杂度为O。

**d.在strategy\_2()中首先根据输入的期待出发时间进行判断，若在今天期待出发时间之后没有线路，则自动切换到明天选择最短时间的路线。再根据期待出发的时间在算法中一段一段地计算最少时间，还要考虑在一个城市停留等待下一趟车的时间，到一个城市的到达时间为下一趟路线的期待出发时间，这样才能保证为最短的时间。**

(3) strategy\_3()函数

strategy\_3()函数为限定时间最短策略，根据用户输入的限定时间利用迪杰斯特拉算法、深度优先搜索和回溯法选择在限定时间内的最少费用。

**a.算法思想**

**先使用一次迪杰斯特拉算法，计算最少费用，判断此路线的总时间是否在限定时间内且按顺序经过了必经城市，若满足则直接输入，若不满足则进行深度优先搜索和回溯法的函数中。**

**从输入的出发城市出发，遍历所有能到达的城市，若能使从出发城市到某城市的费用，加上某城市到下一城市的费用小于一开始出发城市到此城市的费用，或者时间更少，则继续搜索下去。当超过限定时间或者到达终点城市时没有经过必经城市的时候回溯，最后搜索到到达终点城市且经过所有必经城市且在限定时间之内的最少费用策略。**

**b.算法特点**

**使用该算法能保证问题的正确的解，但是由于是深度优先搜索所有的城市，算法的时间复杂度很高，计算量极大。先使用一次迪杰斯特拉算法，若能满足条件，则能很快的得出线路，这使没有途径城市的情况下的程序运行速度大大提高。虽然加上了回溯的条件使满足不了条件时及时停止搜索，以提高算法的性能，但是在限定时间较高的情况下，程序运行的时间还是较长。**

**3.图形化模块**

根据定时器响应周期性事件或者按键按下事件。

在周期性事件中，ShowTime()函数每过一小时（系统时间10s为一小时）在界面上显示当前时间，file()函数则是每小时向文件输出所有旅客的当前状态。pictureout()函数其实是对键盘模块中的screenout()函数进行了输出修改，改成了图形化的输出，每小时在图形化界面上输出旅客当前状态。

在按键按下事件中，则对鼠标点击或者键盘输入进行响应，点击“添加旅客”图标后，guestcome()函数进行添加旅客操作，guestcome()函数中的Input()函数接收用户在界面上输入的字符串，然后根据用户在购票过程中的输入调用键盘模块中strategy\_1()、strategy\_2()、strategy\_3()三个策略函数后为用户设计旅行线路，然后输出至图形化界面上。点击“查询状态”图标后，check()函数对旅客进行查询判断，然后在周期性事件中输出旅客当前状态。点击“关闭”图标后关闭图形界面。